Лабораторна робота №10

Використання у проекті C++ модулів на асемблері

Мета: Навчитися створювати програми на C++ з використанням модулів на асемблері.

Завдання:

1. Створити у середовищі MS Visual Studio проект C++ з ім'ям **Lab10**.
2. Написати вихідний текст програми згідно варіанту завдання. У проекті мають бути такі файли вихідного тексту:
   - головний файл: **lab10.cpp**
   - файли двох модулів на асемблері: **module.asm** та **longop.asm**.
3. У цьому проекті кожний модуль може окремо компілюватися.
4. Скомпілювати вихідний текст і отримати виконуємий файл програми.
5. Перевірити роботу програми. Налагодити програму.
6. Отримати результати – кодовані значення чисел згідно варіанту завдання.
7. Проаналізувати та прокоментувати результати, вихідний текст та машинний код програми.

Теоретичні відомості

Одним з підходів до створення програм на мовах високого рівня, зокрема на мові C++, є використання модулів, написаних на асемблері. У таких модулях можуть міститися процедури, які будуть викликатися з інших модулів, написаних мовою C++. Наприклад, у проекті Lab10 у відповідному файлі *.cpp запрограмовані виклики процедур, які містяться у файлах *.asm. У цьому випадку потрібно знати імена та типи цих процедур, типи їхніх аргументів. Така інформація записується у файли заголовків *.h.

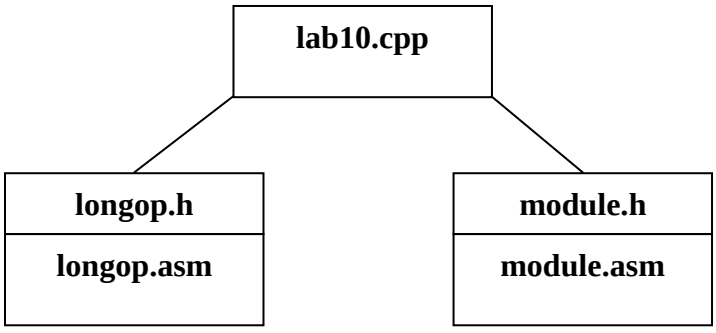

Рис. 1. Взаємодія модулів у програмі на C++

Узгодження виклику процедур

Як написати файл заголовку *.h для процедури на асемблері? Для цього потрібно знати, який спосіб виклику був реалізований у коді процедури. Наприклад, процедури **Add_LONGOP** та **StrHex_MY** були спроектовані так, щоб у модулях на асемблері вони викликалися наступним чином:

```
push offset ValueA
push offset ValueB
push offset Result
push 256
call Add_LONGOP

push offset TextBuf
push offset Result
push 256
call StrHex_MY
```

Як записати прототипи цих процедур мовою C++? Перше, на що звертаємо увагу – ці процедури не повертають значення – мовою C++ це тип **void**. Тоді файл заголовку longop.h для асемблерних процедур **Add_LONGOP** та **Sub_LONGOP**, можна записати, наприклад, у такому вигляді:

```
extern "C"
{
void Add_LONGOP(long bits, long *dest, long *pB, long *pA);
void Sub_LONGOP(long bits, long *dest, long *pB, long *pA);
}
```

Так ми забезпечуємо необхідний порядок запису аргументів у стек – від pA до bits, оскільки для реалізації мови C++ прийнято, що аргументи функцій записуються у стек у порядку справа наліво.
Процедура **StrHex_MY** модуля module.asm має три параметри. Файл заголовку module.h може бути, наприклад, таким:

```
extern "C"
{
void StrHex_MY(long bits, long *src, char *dest);
}
```

Процедури, написані на асемблері, будуть викликатися як звичайні функції C++. Це повинен забезпечити компілятор.

Окрім порядку запису аргументів (параметрів) необхідно визначити – як повинен відновлюватися стек. Процедури, які розглянуті вище, відновлюють стек самі наприкінці відпрацювання свого виклику. Так роблять процедури, написані згідно конвенції, наприклад, **stdcall**. У програмах на C++ використовується інша конвенція виклику – **cdecl**, згідно з якою процедура не

буде сама звільнювати стек від параметрів – це буде робитися після відпрацювання процедури та виходу з неї.

Одним з підходів розробки модулів процедур на асемблері є виконання вимог конвенції виклику **cdecl**, якщо планується використовувати ці процедури у проектах C++. Для цього, у тексті кожної зовнішньо викликаємої процедури замість

ret Кількість байтів для параметрів

повинно бути просто **ret**. Наприклад, для процедури, якій передається 3 параметри через стек, потрібно звільняти стек не командою **ret 12** всередині процедури:

```
StrHex_MY proc
   push ebp
   mov ebp,esp
   . . .              ;решта програмного коду процедури

   pop ebp
   ret 12             ;замість цього повинно бути просто ret
StrHex_MY endp
```

а інструкцією, яка виконується після завершення роботи процедури:

```
push offset TextBuf
push offset Result
push 256
call StrHex_MY
add esp, 12           ;звільнення стеку від параметрів
```

Це інструкція додавання до вказівника стеку (вмісту регістру ESP) відповідного значення. У програмах на C++ таку інструкцію у машинний код автоматично записує компілятор.

Синтаксис процедур на асемблері

Розглянемо вихідний текст на асемблері процедури StrHex_MY. У дещо спрощеному варіанті він має такий вигляд:

```
StrHex_MY proc
   push ebp                ;пролог
   mov ebp,esp

   mov ecx, [ebp+8]        ;кількість бітів джерела даних
   mov esi, [ebp+12]       ;адреса числа
   mov ebx, [ebp+16]       ;адреса буфера результату
```

```
   . . .                    ;решта програмного коду процедури

   pop ebp                  ;епілог
   ret
StrHex_MY endp
```

Можна у першому рядку визначення цієї процедури записати імена та типи параметрів:

```
StrHex_MY proc bits:DWORD, src:DWORD, dest:DWORD
```

Тоді для читання значень параметрів буде зручніше використовувати імена параметрів:

```
mov ecx, bits
mov esi, src
mov ebx, dest
```

хоча ніхто не забороняє для цього прямо вказувати адреси у стеку – тільки треба їх знати:

```
mov ecx, [ebp+8]          ;адреса значення параметру bits
mov esi, [ebp+12]         ;адреса значення параметру src
mov ebx, [ebp+16]         ;адреса значення параметру dest
```

Суттєвим моментом є також те, що у вихідному тексті цієї процедури не треба записувати пролог

```
   push ebp
   mov ebp,esp
```

та епілог

```
   pop ebp
```

– компілятор додасть їх у машинний код сам. Таким чином, вихідний текст процедури StrHex_MY матиме наступний вигляд:

```
StrHex_MY proc bits:DWORD, src:DWORD, dest:DWORD
   mov ecx, bits            ;кількість бітів джерела даних
   mov esi, src             ;адреса числа
   mov ebx, dest            ;адреса буфера результату

   . . .                    ;решта програмного коду процедури

   ret
StrHex_MY endp
```

Такий синтаксис можна використати для інших процедур.

Порядок виконання роботи та методичні рекомендації

1. У середовищі MS Visual Studio створіть новий проект C++ Win32 з ім'ям **Lab10.** Процес створення такого проекту розпочинається у такому вікні (рис. 2)

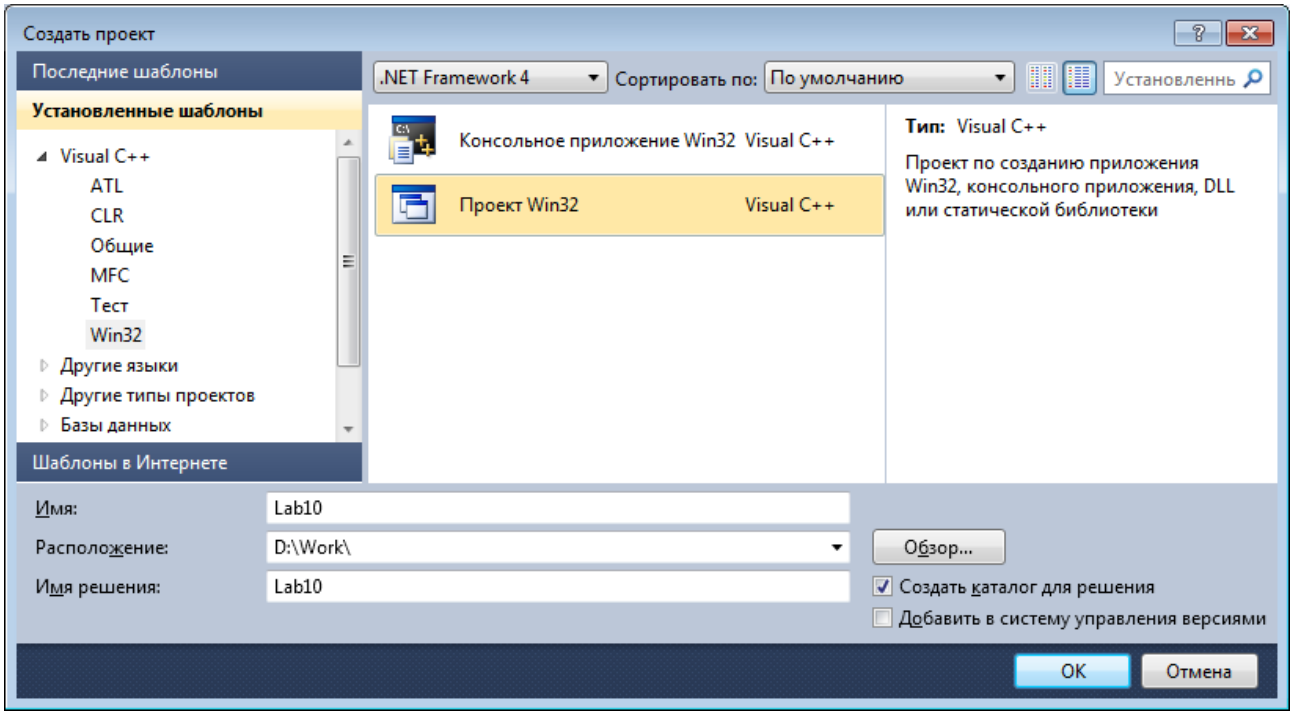

Рис. 2. Початок створення нового проекту

Далі показується вікно з основними параметрами (рис. 3). У цьому вікні змінювати нічого не потрібно (на відміну від проектів попередніх лаб. робіт):

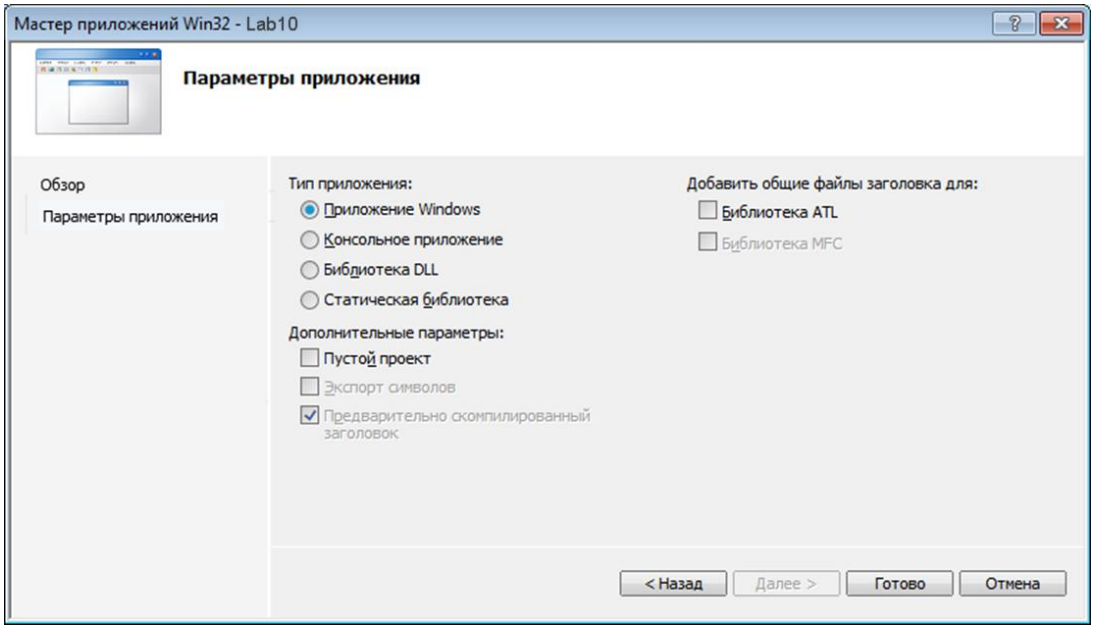

Рис. 3. За умовчанням – те, що потрібно

Після натискування кнопки "Готово" розпочнеться автоматичне створення основних файлів проекту. По завершенню середовище Visual Studio перейде у відповідний стан. Будуть показані вікно головного файлу програми та вікно списку робочих файлів проекту – вікно "Обозреватель решений" (рис. 4):

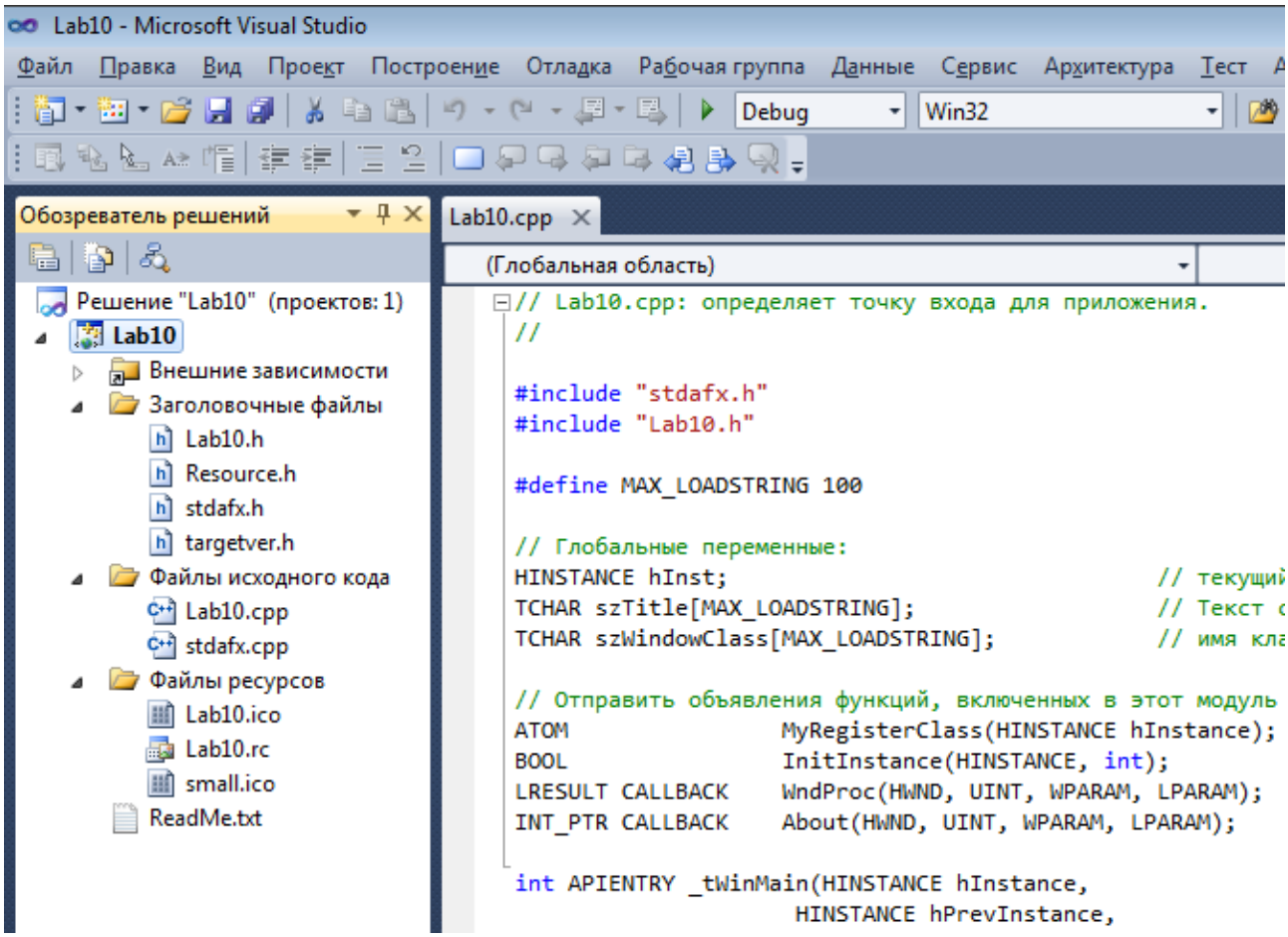

Рис. 4. Файли новоствореного проекту

У вікні головного файлу буде показаний вихідний текст файлу *.cpp, який автоматично створило середовище MS Visual Studio C++.

2. Налаштування однобайтового кодування символів текстів. При роботі з рядками символів, наприклад, при відображенні результату за допомогою MessageBox, в цій лабораторній роботі буде використовуватися однобайтове кодування символів. Саме таке кодування використовується у відомій по попереднім лабораторним роботам процедурі StrHex. Однобайтове кодування символів простіше та зручніше для даної лабораторної роботи. Проте при створенні проекту C++ Win32 автоматично встановлюється Unicode. Щоб змінити набір символів, виберіть меню "Проект – Властивості". З'являється діалогове вікно сторінок властивостей проекту (рис.5). Виберіть "Свойства конфигурации – Общие".

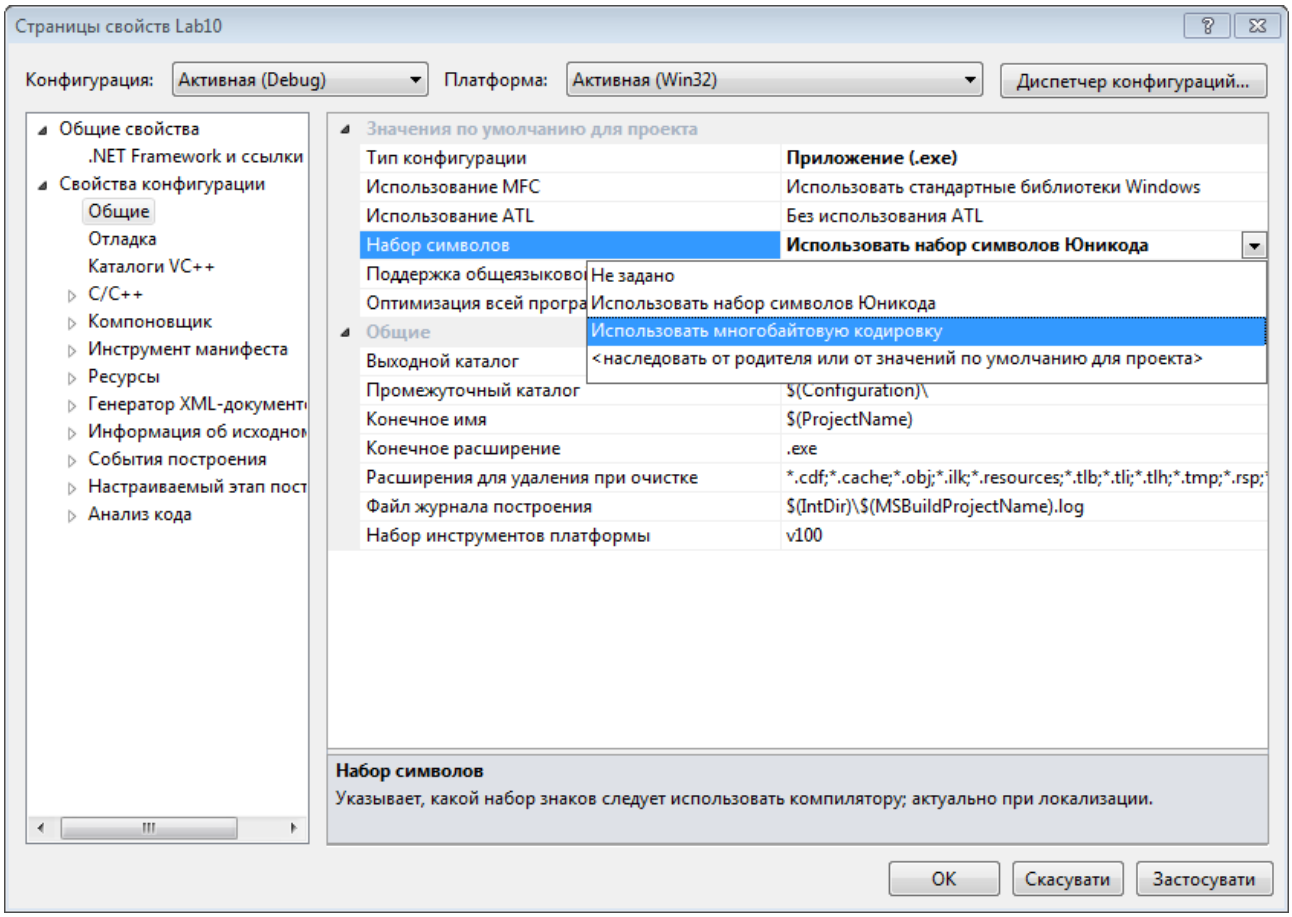

Рис. 5. Вказування однобайтового кодування символів замість Unicode

У рядку "Набор символов" замість "Использовать набор символов Юникода" виберіть "Использовать многобайтовую кодировку".

3. Налаштування підтримки мови асемблеру у проекті.

Таке налаштування повинне бути вже знайоме студентам по попереднім лабораторним роботам цього курсу.

Потрібно вказати курсором у вікні "Обозреватель решений" назву проекту (Lab10) і натиснути праву кнопку миші. У спливаючому меню треба вибрати пункт "Настройки построения. . ." (рис. 6).

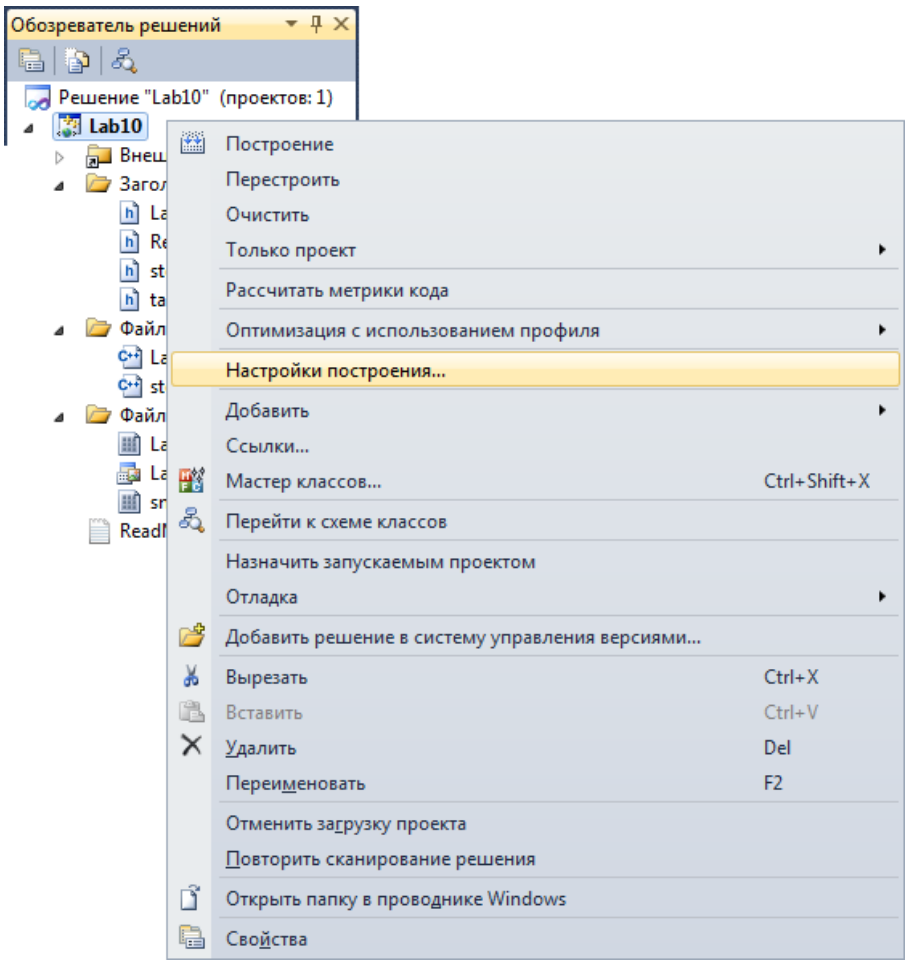

Рис. 6.

Далі з’явиться діалогове вікно (рис. 7):

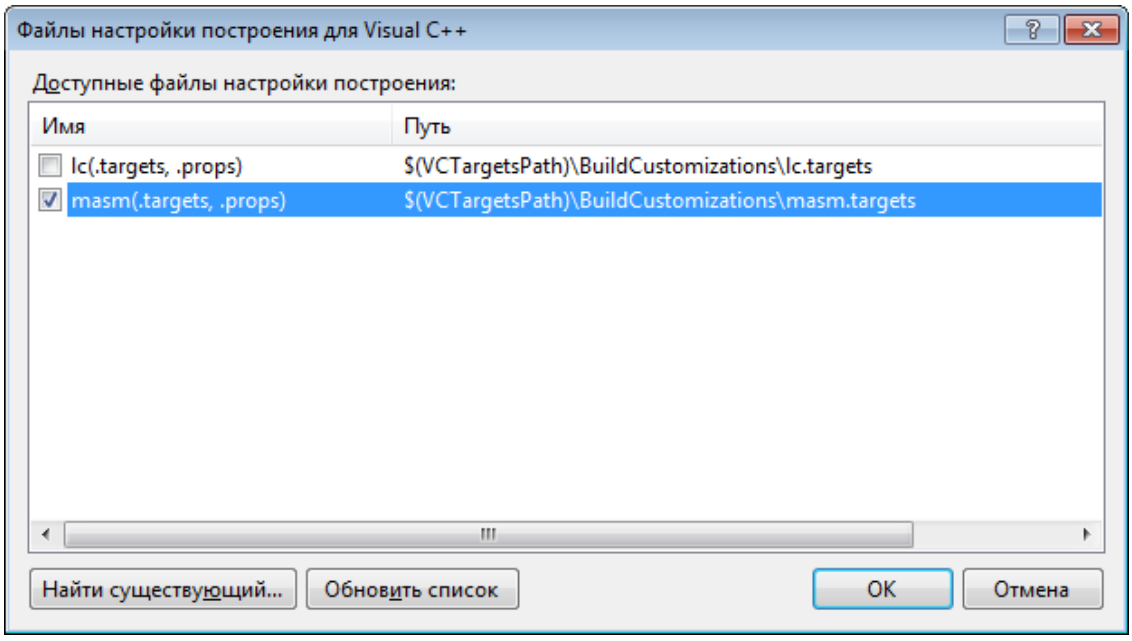

Рис. 7.

Виберіть пункт "masm", та зробіть позначку у квадратику. Натисніть ОК.

4. Додавання модулів у проект.

Потрібно додати файли **longop.asm** та **module.asm** (можна використати файли попередніх лаб. робіт) у розділ "Файлы исходного кода" (рис. 8):

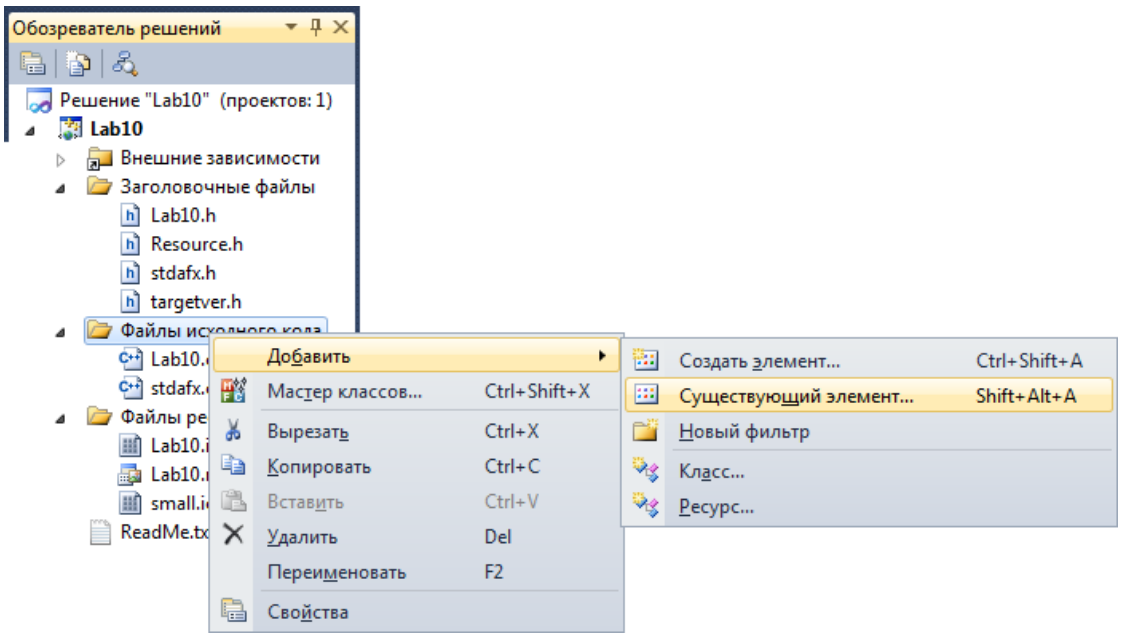

Рис. 8. Додавання файлів вихідного тексту у проект

Потім відкрити вихідні тексти та відредагувати згідно завданню (рис. 9):

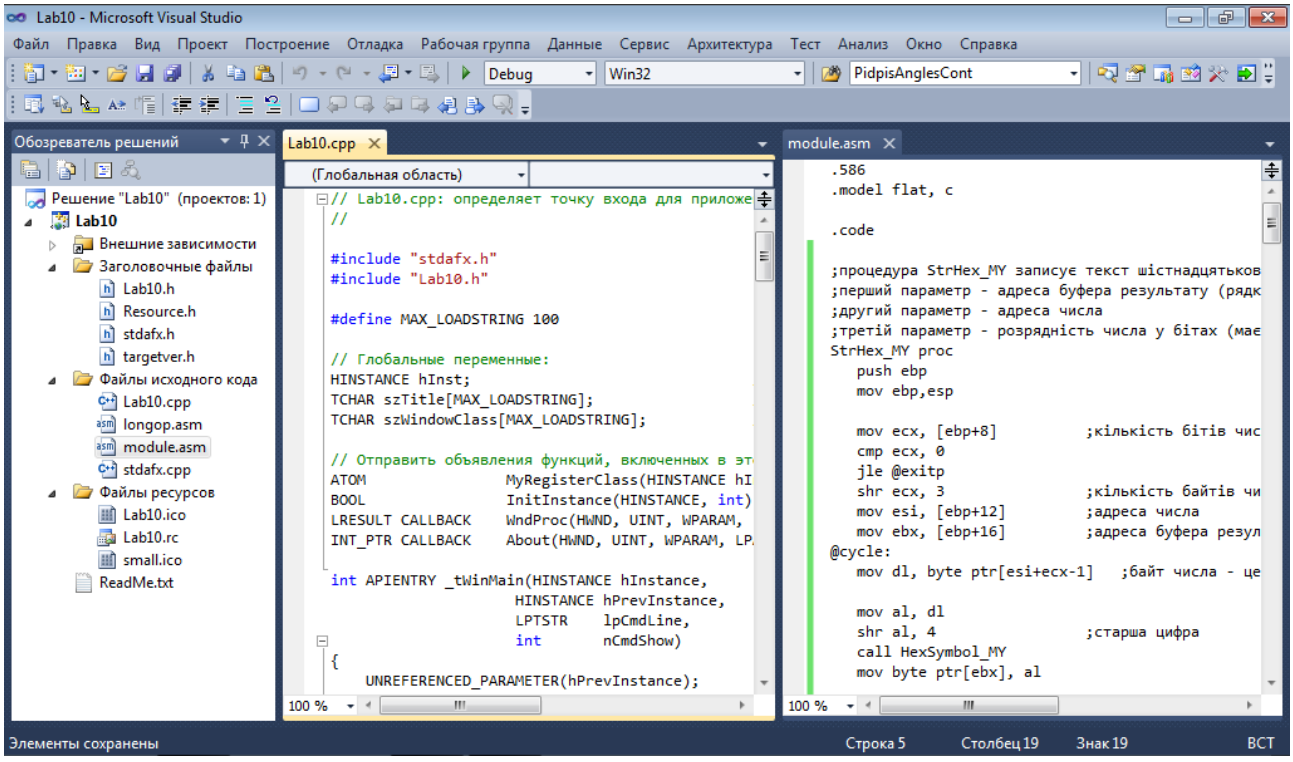

Рис. 9. Проект Visual Studio C++ з модулями на асемблері

5. Редагування меню головного вікна програми.

Для цього у вікні "Обозреватель решения" треба вибрати головний файл ресурсів – для проекту Lab10 це буде файл **Lab10.rc** (рис. 10):

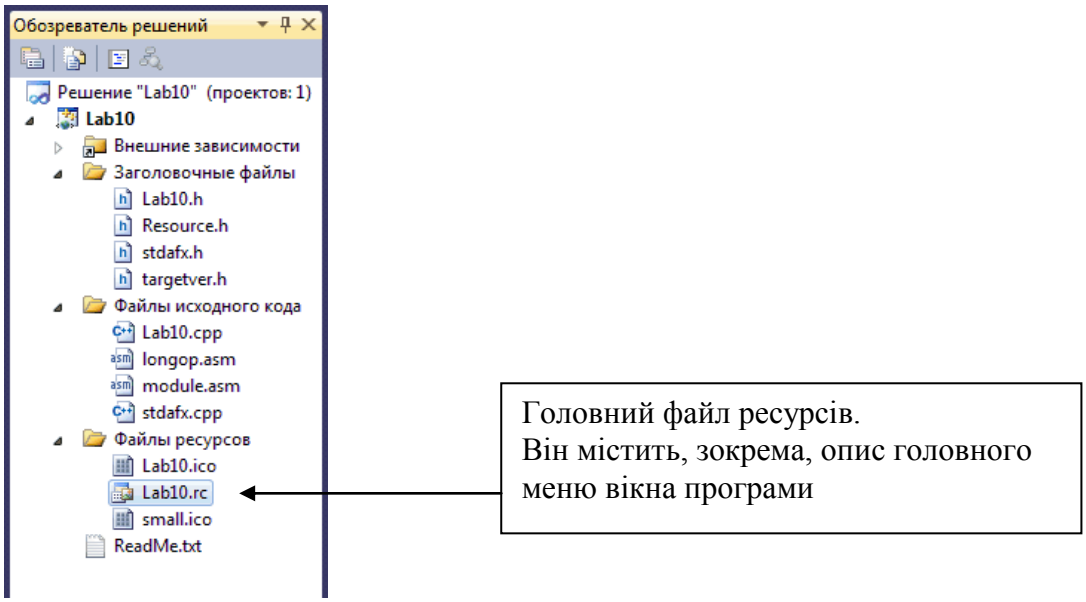

Рис. 10. Головний файл ресурсів

Двічі клацніть лівою кнопкою миші на Lab10.rc, а потім у вікні ресурсів розкрийте пункт меню і двічі клацніть на IDC_LAB10. Повинен з’явитися редактор ресурсів для головного меню. У редакторі ресурсів додати до меню пункт "Виконати – Арифметика".

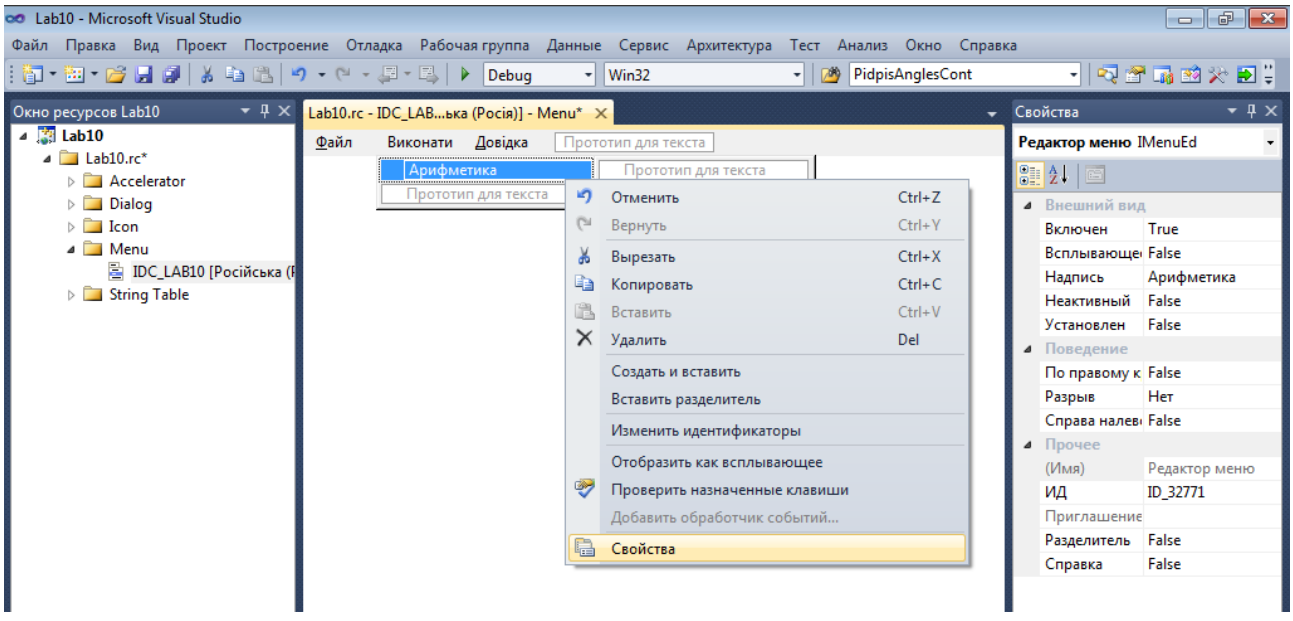

Рис. 11. Редактор ресурсів, викликаний для розробки головного меню

Потрібно узнати ID цього нового пункту меню Про це можна дізнатися, якщо клацнути правою кнопкою миші на цьому пункті і далі вибрати пункт "Свойства" – у вікні "Свойства" пункт ИД вказуватиме значення ID_32771, як проілюстровано на рис. 10. Зазвичай для першого нового пункту меню автоматично створюється символічна константа з ім'ям ID_32771, для наступного ID_32772, і так далі.

6. Перша компіляція проекту.

Виконується через меню "Отладка – Начать отладку". У разі успішної компіляції та виклику програми, повинно з'явитися головне вікно програми з потрібним меню (рис. 11).

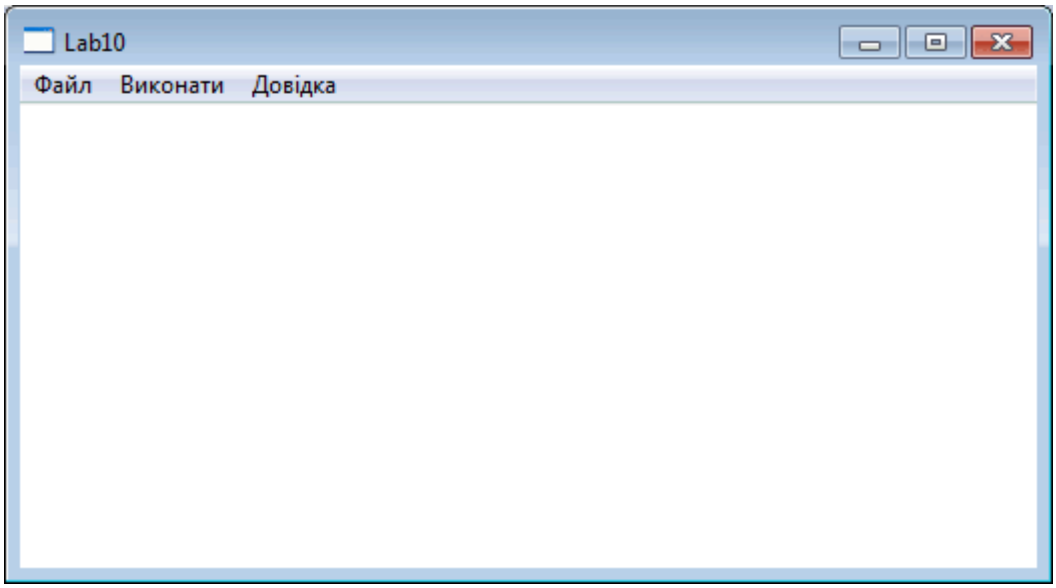

Рис. 11. Вигляд головного вікна програми Lab10

Закрийте програму. Перший крок розробки програми виконано – був започаткований інтерфейс користувача і виконана випробувальна компіляція першої стадії проекту. Потрібно продовжувати роботу над проектом.

7. Редагування вихідних текстів. Потрібно додати у головний файл *.cpp посилання на файли заголовків модулів longop та module. Це робиться за допомогою директив include:

```
#include "longop.h"
#include "module.h"
```

Крім того потрібно додати у функцію головного вікна програми обробку повідомлення про вибір пункту меню. Обробку цього повідомлення

запрограмувати як виклик функції, яку можна назвати, наприклад, ім’ям MyWork1.

```
// Фрагмент функції головного вікна програми
        switch (message)
        {
        case WM_COMMAND:
           wmId    = LOWORD(wParam);
           wmEvent = HIWORD(wParam);
           switch (wmId)
           {
           case ID_32771:         // Вибір пункту меню
              MyWork1(hWnd);
              break;
```

Функцію MyWork1 потрібно оголосити – рядок оголошення запишіть у початку тексту головного файлу *.cpp поруч із оголошенням інших функцій.

Текст визначення функції MyWork1 розмістіть наприкінці тексту головного файлу *.cpp. Потрібно запрограмувати цю функцію згідно з варіантом завдання.

Нижче наданий приклад програмування роботи з процедурами асемблерних модулів, які викликаються як звичайні функції C++.

```
void MyWork1(HWND hWnd)
{
long oA[8]={0x80000001,0x80000001,0x80000001,0x80000001,
            0x80000001,0x80000001,0x80000001,0x80000001};
long oB[8]={0x80010001,0x80020001,0x80030001,0x80040001,
            0x80050001,0x80060001,0x80070001,0x80080001};
long result[8];
char TextBuf[256];

Add_LONGOP(256, result, oB, oA);
StrHex_MY(256, result, TextBuf);
MessageBox(hWnd, TextBuf, "Результат A+B", MB_OK);

Sub_LONGOP(256, result, oB, oA);
StrHex_MY(256, result, TextBuf);
MessageBox(hWnd, TextBuf, "Результат A-B", MB_OK);
}
```

8. Редагування вихідних текстів модулів на асемблері.

У модулі **longop.asm** запрограмувати на асемблері процедури додавання, віднімання та множення чисел підвищеної розрядності. Також у модулі **module.asm** запрограмувати функцію StrHex_MY.

Відредагувати у файлах **longop.asm** та **module.asm** вихідні тексти процедур так, щоб процедури були придатні для виклику у програмах на C++. Ці процедури будуть викликатися у файлі lab10.cpp як функціїї C++.

Шаблон тексту процедур **Add_LONGOP** та **Sub_LONGOP**:

```
;Процедура Add_LONGOP: результат = A + B
;bits - розрядність даних (біт)
;dest - адреса результату
;pA - адреса A
;pB - адреса B
Add_LONGOP proc bits:DWORD, dest:DWORD, pB:DWORD, pA:DWORD
   . . .                    ;решта програмного коду процедури
   ret
Add_LONGOP endp

;Процедура Sub_LONGOP: результат = A - B
;bits - розрядність даних (біт)
;dest - адреса результату
;pA - адреса A
;pB - адреса B
Sub_LONGOP proc bits:DWORD, dest:DWORD, pB:DWORD, pA:DWORD
   . . .                    ;решта програмного коду процедури
   ret
Sub_LONGOP endp
```

Аналогічно можна написати текст процедури **Mul_LONGOP** – множення чисел підвищеної розрядності.

Шаблон тексту процедури StrHex_MY:

```
;Процедура StrHex_MY записує шістнадцятковий код числових даних
;bits - розрядність даних у бітах
;src  - адреса даних
;dest - адреса буфера результату (рядка символів)
StrHex_MY proc bits:DWORD, src:DWORD, dest:DWORD
   . . .                    ;решта програмного коду процедури
   ret
StrHex_MY endp
```

Аналогічно текст процедури, яка виводить десятковий код чисел підвищеної розрядності.

Потрібно також записати файли заголовків – **longop.h** та **module.h**, які містять оголошення процедур, які викликаються з відповідних модулів.

9. Компіляція, налагодження програми.

Основні рекомендації по налагодженню програм у середовищі MS Visual Studio були вже розглянуті у методичних вказівках до попередніх лабораторних робіт.

10. Отримання та аналіз машинного коду. Для цього вказати потрібну точку зупинки і при виконанні програми у режимі налагоджування відкрити вікно дизасемблювання. Виділити та скопіювати у текстовий редактор, наприклад, Word фрагменти машинного коду, які стосуються процедур та функцій, запрограмованих при виконанні роботи. На машинному коді показати послідовність переходів при викликах процедур, інструкції передачі параметрів та звільнення стека.

11. Отримання результатів роботи.

Викликати програму, вибрати пункт меню "Арифметика". У діалогових вікнах прочитати коди результатів виконання операцій, наприклад (рис. 12):

Рис. 12. Результат додавання 256-бітових чисел

Варіанти завдання

Номер варіанту (N) згідно списку студентів у журналі.

Кожному студенту необхідно запрограмувати на C++ обчислення арифметичних виражень шляхом викликів процедур з модулів на асемблері. Результати відображати у шістнадцятковому та десятковому кодах.

№ варіанту	Вираження	Розрядність
1	AB + C – D	512
2	(A+B) (C–D)	544
3	A^2+B^2 – 1	576
4	2(AB+C)	608
5	3 – AB + C	640
6	AB + CD – 1	672
7	A^2 + B – 5	704
8	3A^2 – B + 4	736
9	(1 – A)(B + C)	768
10	(A – B)(C + 3)	800
11	A (B + 1) (C + 2)	832
12	A (B – 1) (C – 2)	864
13	2 – A(B+C)	896
14	2 + A(B – C)	928
15	2A + 3B – C	960
16	3A – 4B + C	992
17	(A – 1) (2B + 3C)	1024
18	A^2 + B^2 – C^2	128
19	A^2 – B^2 + 4	160
20	A^2 – B + C^2	192
21	A^3 + B – 1	224
22	A^3 + B^3 – 1	256
23	A^2(B – C) + 1	288
24	A + B^2 + C^2 – 1	320
25	A + B^2 – C^3	352
26	A(1 + B) – CD	384
27	A^3 + B^2 - C	416
28	50A + 10B – 3C	448
29	29 + A(B – C)	480
30	30 – A^2 + B^2	512

Зміст звіту:

1. Титульний лист
2. Завдання
3. Роздруківка вихідного тексту програми:
   - вихідних текстів модулів на асемблері – повністю
   - головного файлу lab10.cpp – частково, тільки текст, який був доданий для виконання завдання
4. Роздруківка машинного коду – частково, тільки фрагментів, які відповідають процедурам та функціям, розробленим при виконання завдання
5. Роздруківка результатів виконання програми
6. Аналіз, коментар результатів, вихідного тексту та машинного коду
7. Висновки

Контрольні питання:

1. Як налагодити підтримку асемблеру у проекті Visual C++?
2. Як у проекті вказати робочий набір символів multibyte замість Unicode?
3. Що таке конвенція cdecl?
4. Як відредагувати меню програми?
5. Який машинний код додає компілятор при виклику процедури?
6. Чому аргумент функції Func(char *dest), який мовою C++ має тип char*, при програмуванні на асемблері повинен мати тип DWORD?